TRANSPORTATION ANALYSIS SIMULATION SYSTEM (TRANSIMS)

Version: **TRANSIMS-LANL-1.0**

VOLUME 2 – SOFTWARE
PART 2 – SELECTORS

28 May 1999

LA-UR 99-2575

TRANSIMS

Version: TRANSIMS-LANL-1.0

VOLUME 2 – SOFTWARE
PART 2 – SELECTORS

28 May 1999

LA-UR-99-2575

The following persons contributed to this document:
C. L. Barrett*
R. J. Beckman*
K. P. Berkbigler*
K. R. Bisset*
B. W. Bush*
S. Eubank*
J. M. Hurford*
G. Konjevod*
D. A. Kubicek*
M. V. Marathe*
J. D. Morgeson*
M. Rickert*
P. R. Romero*
L. L. Smith*
M. P. Speckman**
P. L. Speckman**
P. E. Stretz*
G. L. Thayer*
M. D. Williams*

* Los Alamos National Laboratory, Los Alamos, NM 87545
** National Institute of Statistical Sciences, Research Triangle Park, NC

Acknowledgments

This work was supported by the U. S. Department of Transportation (Assistant Secretary for Transportation Policy, Federal Highway Administration, Federal Transit Administration), the U. S. Environmental Protection Agency, and the U. S. Department of Energy as part of the Travel Model Improvement Program.

CONTENTS

1. OVERVIEW

The process of iterative feedback is a key distinguishing feature of TRANSIMS. It first allows the overall computational system to reflect *learned* behavior within the simulated population represented (i.e., the ability of humans to learn from day-to-day experiences in order to avoid congestion, etc.). It also provides a way to simulate intelligent responses to information that may be provided ultimately by intelligent transportation systems technologies.

The *Selector* is the TRANSIMS framework component that controls the iterative process. A typical TRANSIMS study involves repeated iteration between components such as the Activity Generator, Route Planner, and Traffic Microsimulator. There is no single, *standard* selector component, however, because different study designs involve different iteration schemes. A variety of selectors have uses in different studies or other contexts. In this document, we will describe a single, canonical selector. Figure 1 illustrates where the Selector resides within the TRANSIMS framework.

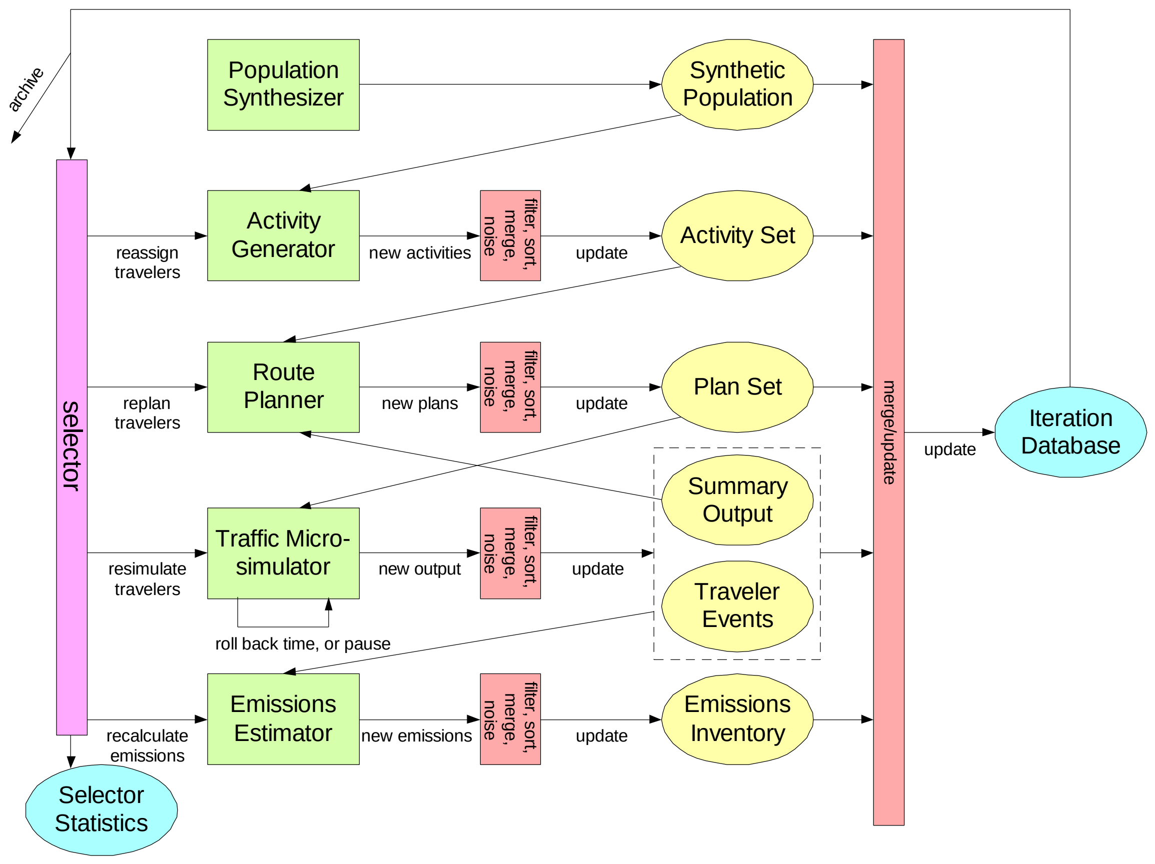

Figure 1. Location of the Selector within a typical TRANSIMS experimental design.

The user can prepare an *iteration script* to control the whole process of iteration. The script uses special control commands specifically developed for this iterating of TRANSIMS components. It allows the user to filter results, run repeated iteration, establish stopping criteria, and perform a host of other operations that make the analyst's job less manpower intensive.

During each iteration, the iteration script controlling the current study typically invokes a selector. (The script might even use a different selector for each iteration in a study.) When a selector runs, it usually will do the following:

- Read information about the travelers from the iteration database.
- Examine each traveler and decide whether to
  - regenerate his/her activities using the activity generator,
  - choose a new route between his/her existing activities using the Route Planner, or
  - retain his/her existing activities and the planned route between them.
- Write the selections made for each traveler into data files that can be read by the Activity Generator and/or Route Planner when they are executed.
- Summarize the selections made and the current state of the system into a selector statistics data file.

Figure 2 illustrates a selector's decision-making process.

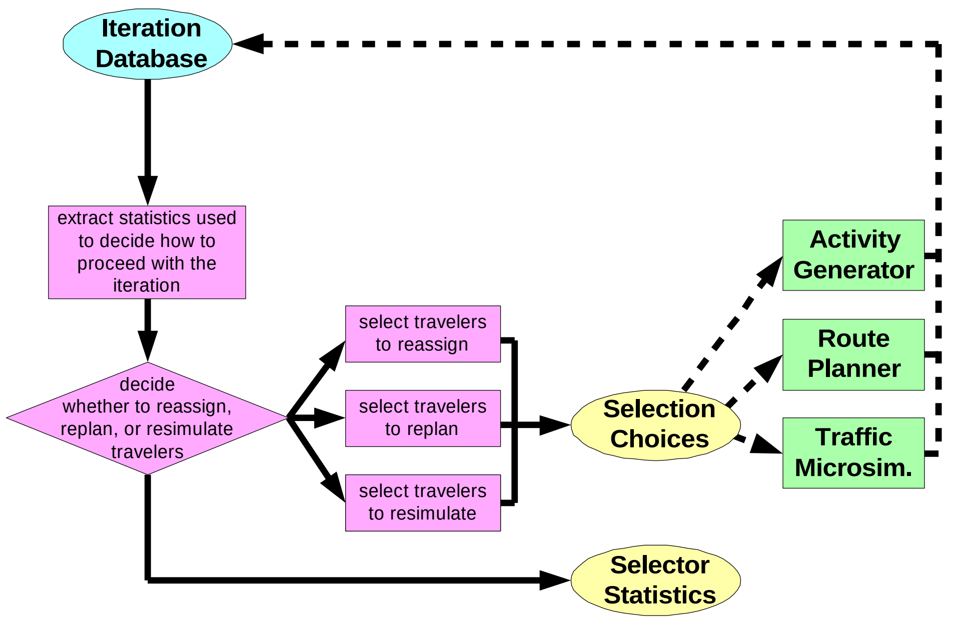
Iteration Database
extract statistics used to decide how to proceed with the iteration
decide whether to reassign, replan, or resimulate travelers
select travelers to reassign
select travelers to replan
select travelers to resimulate
Selection Choices
Activity Generator
Route Planner
Traffic Microsim.
Selector Statistics

Figure 2. Typical Selector logic.

After the Selector completes the selection process for all of the travelers, the Activity Generator, Route Planner, or Traffic Microsimulator runs to calculate the updated activity set, plan set, or microsimulation output files, respectively, according to the decisions made by the Selector. The iteration script will reinvoke a selector again at the start of the next iteration in the study. Figure 3 shows examples of four possible progressions, as determined by the Selector.

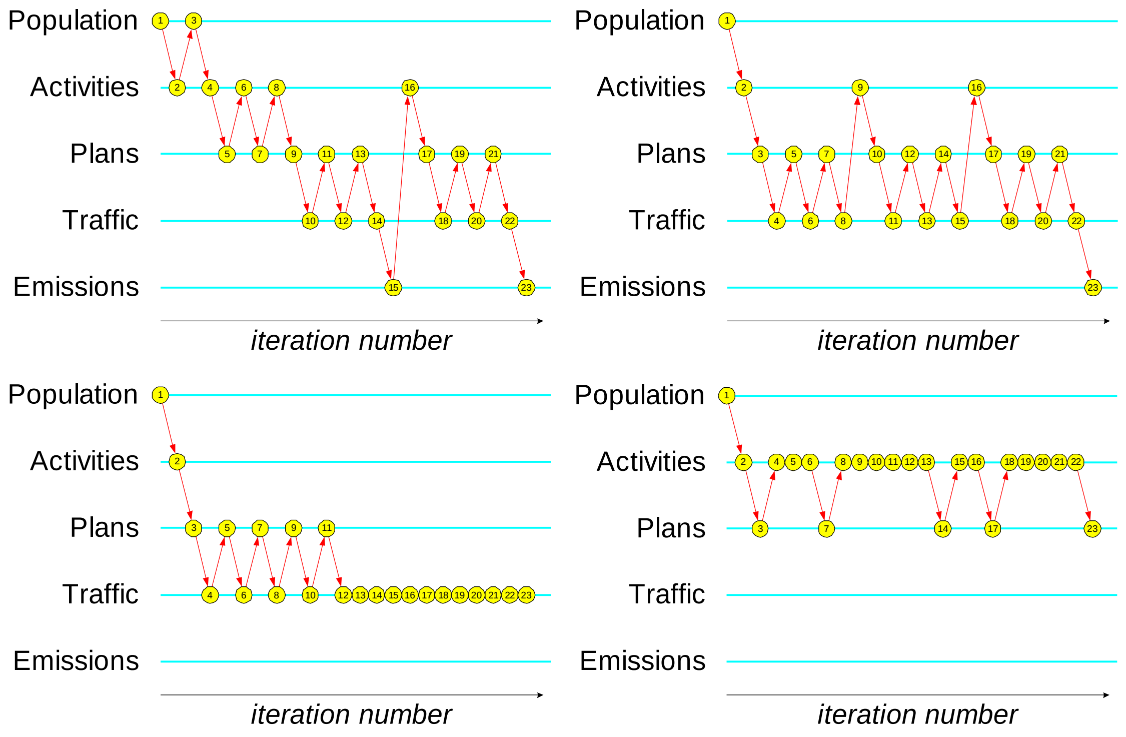

Figure 3. Four example iteration progressions.

The major input to the Selector is the *iteration database*. It contains a summary history of each traveler's attributes, expectations, and experiences during the iterations within a study. The Selector uses these data items to make its selection decisions. *Attributes* represent quasi-static information about travelers like their age, income, gender, or profession. *Expectations* encompass information such as how long a traveler expects to travel between two of his/her activities based on the route between them generated by the Route Planner. *Experiences* compose information extracted from detailed Traffic Microsimulator output—for instance, the actual travel time realized in the microsimulation between two activities. The analyst may choose which attribute, expectation, and experience data reside in the iteration database for a particular study. These data form the universe of information readily available to the Selector; additional data from activity sets, plan sets, and microsimulation output might also be used by some selector implementations, however. The right side of Figure 4 shows this data flow into the Selector.

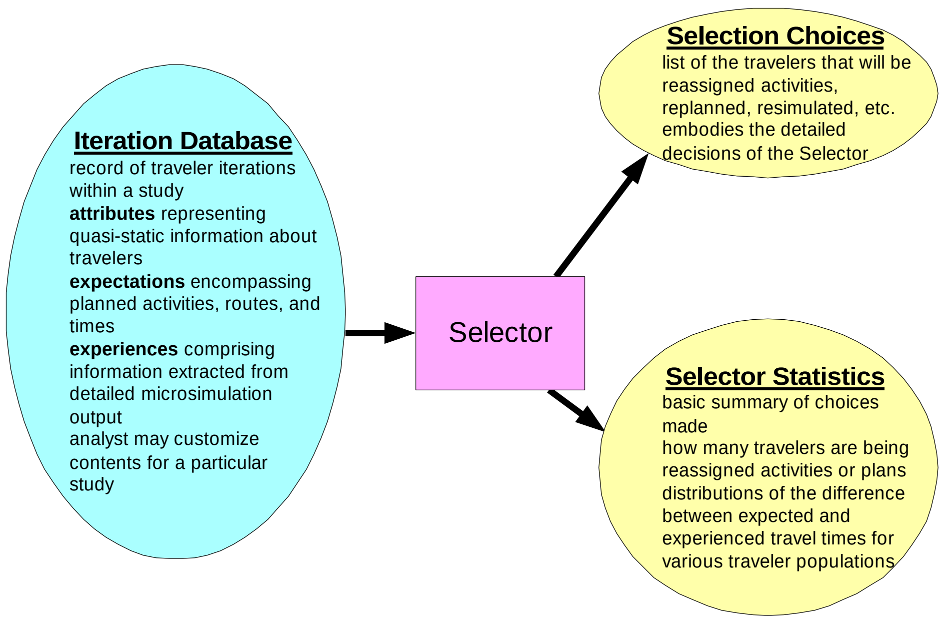
Iteration Database
record of traveler iterations within a study
attributes representing quasi-static information about travelers
expectations encompassing planned activities, routes, and times
experiences comprising information extracted from detailed microsimulation output
analyst may customize contents for a particular study
Selector
Selection Choices
list of the travelers that will be reassigned activities, replanned, resimulated, etc.
embodies the detailed decisions of the Selector
Selector Statistics
basic summary of choices made
how many travelers are being reassigned activities or plans
distributions of the difference between expected and experienced travel times for various traveler populations

Figure 4. Typical Selector data flow.

The Selector also has two principal outputs: selector statistics and selection choices. The *selection choices* files simply list the travelers that will be reassigned activities, replanned, resimulated, etc.; these files embody the detailed decisions of the Selector. The *Selector statistics* provide a basic summary of the choices a selector makes, e.g., how many travelers are being re-planned, distributions of the difference between expected travel times and experienced travel times for various traveler populations, and the like. The left side of Figure 4 shows this data flow out from the Selector.

2. ALGORITHM

If the SEL_FILL_ITDB configuration key is set to true when the Selector is invoked, the Selector first updates the contents of the iteration database using the latest population, activity, plan, and traveler event data files generated by the Population Synthesizer, Activity Generator, Route Planner, and Traffic Microsimulator, respectively—this is the *merge/update* function shown on the right side of Figure 1. Table 1 summarizes the contents of the iteration database used by the Selector described here.

Table 1. Description of iteration database fields.

Field Description	Description	Source of Data
TRAVID	the ID of the traveler	population file
HOUSEID	the ID of the traveler's household	population file
TRIPID	the ID of the traveler's trip	plan file
LEGID	the ID of the trip's leg	plan file
DESIRED_ARRIVAL_TIME	the desired arrival time (measured in fractional hours) at the activity	activity file
DESIRED_ARRIVAL_TIME_UB	the upper bound for desired arrival time (measured in fractional hours) at the activity	activity file
DESIRED_ARRIVAL_TIME_A	beta distribution parameter specified in the activity file	activity file
DESIRED_ARRIVAL_TIME_B	beta distribution parameter specified in the activity file	activity file
EXPECTED_ARRIVAL_TIME	the expected arrival time (measured in seconds past midnight) at the activity	plan file
ACTUAL_ARRIVAL_TIME	the actual arrival time (measured in seconds past midnight) at the activity	traveler event file
NUM_STOPS	the number of stops signs the traveler encountered on this leg	traveler event file
TIME_STOPPED	the number of seconds the traveler was stopped in traffic on this leg	traveler event file
TOTAL_DISTANCE	the total distance (measured in meters) traveled on this leg	traveler event file
TOTAL_TIME	the total time (measured in seconds) traveled on this leg	traveler event file
GEOM_DISTANCE	the straight-line (Euclidean) distance (measured in meters) planned for travel on this leg	plan file
MODE_PREF	the traveler's mode preference for this leg (same integers as in activity file)	activity file
EXPECTED_DURATION	the expected duration (measured in seconds) of the current leg	plan file

After the iteration database has been updated, the Selector builds a variety of cost functions: namely,

- *Duration cost:* $c_{duration}(i_{traveler,leg}) = \frac{T_{actual} - T_{expected}}{T_{expected}}$, where T_{actual} is the actual travel time for the trip as realized by the Traffic Microsimulator and $T_{expected}$ is the expected travel time for the leg as estimated by the Route Planner. This measures travel time "frustration."
- *Distance cost:* $c_{distance}(i_{traveler,leg}) = \frac{D_{actual} - D_{geometric}}{D_{geometric}}$, where D_{actual} is the actual distance traveled in the Traffic Microsimulator and $D_{geometric}$ is the point-to-point Euclidean distance between the leg's endpoints. This measures how far out of his or her way the traveler goes.
- *Stopped cost:* $c_{stopped}(i_{traveler,leg}) = \frac{T_{stopped}}{T_{actual}}$, where $T_{stopped}$ is the time stopped in traffic and T_{actual} is the total travel time. This measures the fraction of the time a traveler spends waiting.
- *Late cost:* $c_{late}(i_{traveler,leg}) = A_{desired} - A_{actual}$, where $A_{desired}$ is the arrival time desired by the Activity Generator and A_{actual} is the actual arrival time realized by the Traffic Microsimulator. This measures how late the traveler is for his or her activity.
- *Effective speed cost:* $c_{speed}(i_{traveler,leg}) = \frac{D_{geometric}}{T_{actual}}$, where $D_{geometric}$ is the point-to-point Euclidean distance between the leg's endpoints and T_{actual} is the total travel time. This measures the traveler's effective speed through the network.

Once those cost functions have been built, they are sampled to determine what actions are taken for which travelers: namely,

- Select the fraction $f_{reassign}$ of the travelers uniformly at random, $c_{distance}(i_{traveler,leg})$, for reassignment of activity locations by the activity generator, followed by rerouting by the Route Planner.
- Select the fraction f_{remode} of the travelers with the highest effective speed costs, $c_{speed}(i_{traveler,leg})$, for reassignment of mode preferences by the activity generator, followed by rerouting by the Route Planner.
- Select the fraction f_{retime} of travelers with the highest late costs, $c_{late}(i_{traveler,leg})$, for reassignment of activity times by the Activity Generator, followed by rerouting by the Route Planner.
- Select the fraction $f_{reroute}$ of travelers with the highest duration costs, $c_{duration}(i_{traveler,leg})$, for rerouting by the Route Planner.

The end result of the selection choices is a pair of files containing the list of travelers whose activities must be regenerated (along with what type of regeneration must be done) and the list of travelers whose routes must be replanned.

The iteration script supplied with the Selector ties the iteration process together by running the components, merging files, and creating indexes as needed, and archiving output data into iteration specific directories.

3. USAGE

The Selector takes a single command-line argument, the TRANSIMS configuration file. Table 2 lists the configuration parameters that the Selector uses: these are the various reassignment and rerouting thresholds discussed in the previous section and the location of the iteration database file.

Table 2. Selector configuration file parameters.

Configuration Key	Description
SEL_REROUTE_FRAC	$f_{reroute}$
SEL_RETIME_FRAC	f_{retime}
SEL_REASSIGN_FRAC	$f_{reassign}$
SEL_REMODEFRAC	f_{remode}
SEL_FRUSTRATION_THRESH	f_{remode}
SEL_ITDB_FILE	the iteration database file
SEL_FILL_ITDB	whether to update the iteration database
ACTIVITY_FILE	(index of) activities
PLAN_FILE	(index of) plans
OUT_EVENT_NAME_1	traveler event data created by microsimulation
OUT_DIRECTORY	directory containing output data created by the simulation
ACT_FEEDBACK_FILE	traveler ids and generator command output
ROUTER_HOUSEHOLD_FILE	household ids which need to be rerouted (includes all households with travelers included in the ACT_FEEDBACK_FILE)

4. TUTORIAL

```
   % $TRANSIMS_HOME/bin/Selector config > selector.log
```

After this runs, it will have updated ITDB*_FILE*, if requested. It will also have generated the *ACT*_FEEDBACK*_FILE* and the *ROUTER*_HOUSEHOLD*_FILE*. The standard output contains a cryptic debugging message at the top, followed by value for all of the cost functions. Each cost function's values are separated by a comment line beginning with a '#' character, which offers a brief reminder of what the cost function is.

5. TROUBLESHOOTING

All of the simulation output files must exist, and the SEL_… configuration parameters must be set.

6. ITERATION SCRIPT

6.1 Overview

The purpose of this script is to automate an iteration process. It . . .

- builds and archives the inputs (populations, vehicles, activities, and plans) for each iteration,
- builds the required indexes,
- runs the Traffic Microsimulator
- archives some output (currently animation binary files) from each iteration,
- appends to the iteration database summary statistics for each traveler after each iteration, and
- merges new inputs into previous inputs to prepare for the next iteration.

It can be used with a "clean slate," in which none of the inputs are available, or with previously prepared input files. It maintains separate input archives for different types of populations (e.g., mass transit, freight, transient, and "household population" or "pop"). Input data for each type can be generated differently. For example, transit schedules can be parsed to generate transit driver plans and vehicles, while the router is used to generate plans for members of the household population.

6.2 Algorithm

For each population type, the script executes the following procedures, which are described in more detail below:

1) Build a population
2) Locate a population
3) Build vehicles
4) Build Activities
5) Build Plans

The results of each of these procedures are placed in a subdirectory of the current working directory named *it.<n>*, where *<n>* is the (zero-based) iteration number. This subdirectory is further divided into a subdirectory for each different population type. For example, if the configuration key PLAN_FILE is set to */home/transims/net_plans*, then when the "pop" population type plans are created on iteration 3, they are placed into the file *it.3/pop/net_plans*.

As each result is obtained, it is merged into an index in the "current" subdirectory, which is also subdivided by population type. Thus, in the example above, the new plans would be merged into the index in *current/pop/net_plans*. These indexes contain all of the data from previous iterations that has not been overridden, whereas the ones in *it.<n>* contain only the data generated on the most recent iteration. For efficiency, the plan indexes in the “current” subdirectory are also truncated to the start and end times of the simulation.

Finally, a link is formed between a file in the run directory (current working directory) and the corresponding population-type specific index in the “current” directory. In the example above, *net_plans.trv.idx* would become a link to the file *current/pop/plans.trv.idx*. This is done so that each procedure will have available to it the most recent results of all the previous procedures for that population type.

After every population type has been considered, plan and vehicle indexes are constructed using the indexes in each of the “current” subdirectory’s population types. These contain all of the data from every iteration (except that which has been overridden by later iterations) from all the population types. These indexes are the ones used by the Traffic Microsimulator.

After the Traffic Microsimulator runs, the Selector builds the iteration database and selects travelers for activity regeneration and rerouting. On succeeding iterations, the Route Planner and Activity Generator will use the feedback files to update only the required travelers. Also, the router will use the *OUT_SUMMARY_NAME_1* file created in the previous iteration to obtain link travel time delays.

6.2.1 Build a Population

If necessary (but not for transit), the *BuildPop* procedure calls the *Popgen* executable after removing the POP_BASELINE_FILE for each population type. It moves the POP_BASELINE_FILE it creates into the appropriate iteration’s population type subdirectory. Since *Popgen* is called to build an entire population, not just to modify certain members, the new population file overwrites (via a UNIX link) any previous population in the “current” population type subdirectory, rather than merging into it. A link is also formed between the file in “current” and POP_BASELINE_FILE.

6.2.2 Locate a Population

If necessary (but not for transit), the *LocPop* procedure calls the *Poploc* executable after removing the POP_LOCATED_FILE for each population type. It moves the POP_LOCATED_FILE it creates into the appropriate iteration’s population type subdirectory. Since *Poploc* is called to locate an entire population, not just to modify certain members, the new population file overwrites (via a UNIX link) any previous population in the “current” population type subdirectory, rather than merging into it. A link is also formed between the file in “current” and POP_LOCATED_FILE.

6.2.3 Build Vehicles

If necessary, for the “pop” population type, the *BuildVehicles* procedure calls the *Vehgen* executable. For the “transit” population type, it does nothing, because vehicles will be generated when the transit driver plans are built.

First, the script removes VEHICLE_FILE and its associated indexes, then it generates vehicles. It creates the associated indexes and moves them all to the iteration directory in the appropriate population type subdirectory. Then, since it has generated a complete set of vehicles and not just an update to the previous set, it replaces the vehicle file and indexes in the “current” subdirectory with those in the iteration subdirectory and links VEHICLE file to the ones in the "current" subdirectory.

6.2.4 Build Activities

On the first iteration, this procedure runs the Activity Generator, which processes the entire population file and creates activities in the ACT_FULL_OUTPUT file. On subsequent iterations, it runs the LANL regenerator, which processes only those activities listed in ACT_FEEDBACK_FILE and creates activities in the ACT_PARTIAL_OUTPUT file.

The script moves the output and the traveler and household indexes into the appropriate iteration and population type subdirectory, and merges them into the "current" population type subdirectory. Finally, these indexes are linked into indexes for the file specified by ACTIVITY_FILE.

6.2.5 Build Plans

Based on the activity file and vehicle file generated in earlier steps, the Route Planner creates a plan for each traveler on the first iteration. On subsequent iterations, the file specified by ROUTER_HOUSEHOLD_FILE may exist. If so, the Route Planner will generate routes only for the households indicated in that file.

The Route Planner uses free speed delays on links by default. If the file specified by OUT_DIRECTORY and OUT_SUMMARY_NAME_1 exists, the link-specific time delays given in that file are used instead.

After generating the plan file (which will be temporarily placed in PLAN_FILE), the script moves the plan file and a traveler index to the iteration subdirectory, in the appropriate population type subdirectory. The time-sorted index is not created at this point, because it will not be handled properly by *MergeIndices*. The traveler index is merged into the one in the "current" directory for the appropriate population type. The resulting time index is linked to an index for the file specified by PLAN_FILE.

6.3 Usage

```
   % /bin/sh expt.sh <config-file>
```

where `<config-file>` is the absolute (full) pathname of a configuration file.

6.4 Configuration Keys

Configuration Key	Description
BASELINE_FILE LOCATED_FILE VEHICLE_FILE ACTIVITY_FILE PLAN_FILE	These configuration keys specify base names for the indicated files that are used as starting points for iteration 0. For each of the population types "pop" and "transit", the corresponding type's string is added to the base file name. If the resulting file exists, it will be used. Otherwise, the script will call an appropriate tool to generate it. The last component of the path in each of these values will be used as a filename for the corresponding file in the run directory. A new configuration file will be generated in the run directory with the new file name information overriding the old values. For example, suppose PLAN_FILE is */home/transims/plans*. Then, on the 0th iteration, the script will look for */home/transims/plans.transit*. If it exists, it will be used for the transit driver plans. If not, another executable will be called to generate the plans. This executable will not update */home/transims/plans.transit*. Next, the script will look for */home/transims/plans.pop*. If it does not exist, the Route Planner will be called to generate a plan set. The Traffic Microsimulator will look for the file *plans* in the run directory. This file, generated by the iteration script, will contain all of the plans in both *current/pop/plans* and *current/transit/plans*. This allows the re-use of a particular starting point in an experiment, even if it is very expensive to construct all the files that make up the starting point.
EXPT_NUM_ITER	The number of iterations to perform. Default = 1.

6.5 Troubleshooting

The last component of every file name must be unique or files will be overwritten!